UN CONTADOR CON ESTADÍSTICAS EN LENGUAJE PERL - PRIMERA PARTE

Programación en Perl

Ya hablamos de ASP, aprendimos qué es PHP, pero nunca mencionamos nada acerca de este **NOBLE LENGUAJE**. Perl permite agregar el buscado **DINAMISMO** a las páginas web, realizar consultas y operaciones en **BASES DE DATOS**, y mucho más. En esta nota daremos los primeros pasos con nuestro primer programa Perl.

Martín A. SCANDROLI
masc@rec-inf.com.ar

Programador de Perl, C, diseño web y otras yerbas. Actualmente trabaja en el área de la electromedicina y radiología.

A principios de los noventa, un programador llamado Larry Wall decidió facilitar las cosas para la elaboración de tareas comunes bajo el entorno UNIX. Así decidió crear Perl (*Practical Extraction and Report Language*, Lenguaje Práctico de Extracción y Reportes), un lenguaje noble, sencillo y práctico para quienes se inician, y para aquellos que estaban acostumbrados a programar en C o C++.

Perl es una excelente herramienta para facilitar el procesamiento de grandes volúmenes de información sin sacrificar rendimiento, debido a que basa su propia sintaxis en la mayoría de las herramientas de UNIX. Esta característica hace de Perl un lenguaje ideal para la elaboración de los CGI en un servidor de Internet.

¿Dónde lo puedo utilizar?

Concebido para UNIX, es ahí donde logra su mejor desempeño. Los sistemas DOS no tienen un manejo lo suficientemente bueno de los procesos o de la memoria para lograr el rendimiento ideal de Perl, pero existen emuladores que permiten correr scripts, lo cual resulta de gran utilidad para probar los programas antes de subirlos al servidor.

Programas fuente

Todas las versiones de Perl son de distribución gratuita, e Internet está desbordada de fuentes y recursos disponibles para bajarlos gratuitamente (ver **Para saber más**), pero a la hora de buscar, es una buena idea comenzar con Perl.org **(www.perl.org)**, el sitio oficial de Perl, o con Yahoo! (**ar.dir.yahoo.com/Internet_y_computadoras/Lenguajes_de_programacion/Perl/**).

Es importante aclarar que Perl no actúa ni como compilador ni como intérprete. Es más bien un paso intermedio que de aquí en adelante llamaremos "intérprete de Perl". Nuestros programas deberán ser escritos en ASCII puro, con extensión .pl, luego compilados y ejecutados en el mismo servidor. Para quienes se fanaticen con este lenguaje, ya existen compiladores de la versión 5 para crear sus propios .exe.

Todo un buen programador

Recuerden que los programas deben escribirse con suma claridad. Si bien esto parece obvio, y muchos lo sabemos, son pocos los que siguen esta re-

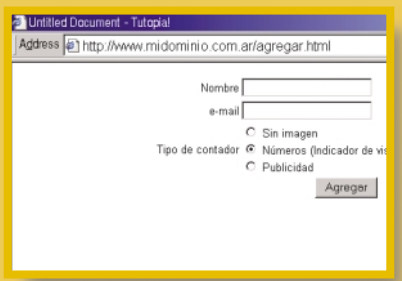


Figura 1. Formulario de ingreso de usuario. Puede tener el aspecto de nuestro sitio, lo importante es llamar correctamente al cgi.

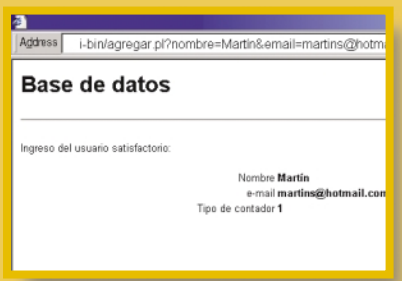


Figura 2. Pantalla de salida. Es importante para confirmar la correcta ejecución del script. También es aconsejable que luzca como nuestro sitio.

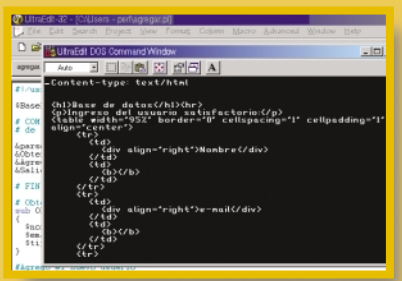
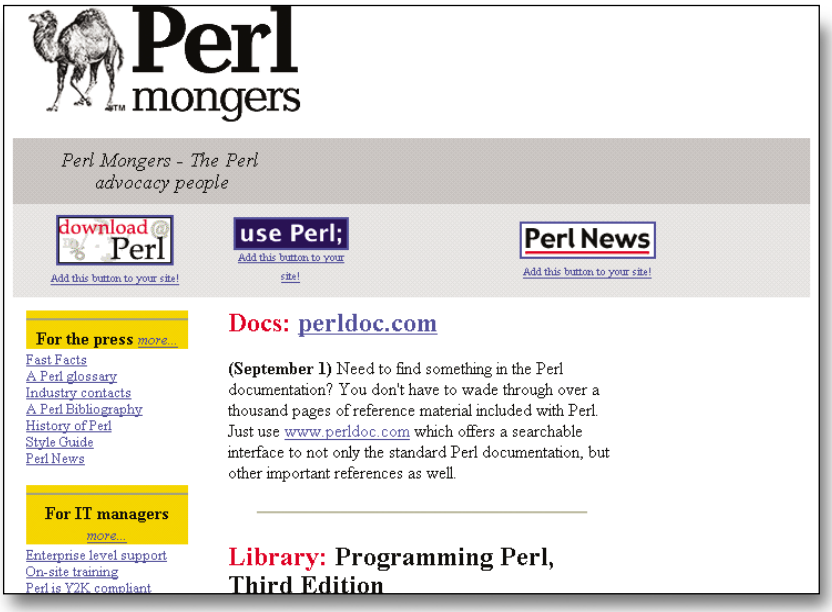


Figura 3. El editor de textos UltraEdit corriendo Perl. Con la línea perl %p%n%e, en la opción [Avanced/ DOS Command], con <F9> ejecutamos nuestros programas.



Uno de los lenguajes más nobles y sencillos para la creación de páginas web. Con diversos caminos para la ejecución de tareas, se ha convertido en una alternativa simple y económica para encarar proyectos en la Web.

gla. Al escribir hay que ser cuidadoso; de lo contrario, no se hallarán ni siquiera los propios errores. Lo ideal es comentar cada función, explicando brevemente qué es lo que hace. En Perl esto se realiza colocando el símbolo **#** delante del texto; de esta forma, el compilador omite lo que se escribió a la derecha hasta que encuentra un **<ENTER>**. Así, cuando queramos modificar o mejorar un programa, y lo retomemos luego de un largo período, nos será más fácil identificar las variables, las funciones y el modo de hacer las cosas. Otra ayuda es la clásica tabulación: cada vez que se abre un bloque (for, while, if, etc.), hay que dejar dos espacios adelante. En resumen, es aconsejable realizar una escritura clara y de fácil entendimiento, tanto para nosotros como para otras personas.

¿Qué necesitamos?

Una de las herramientas más importantes es **Active Perl 5.22** (**www.activestate.com/pw32/**), que servirá para ejecutar y probar los programas en la PC bajo Windows 32. Para los que cuenten con la posibilidad de tener Linux, o subir los programas al servidor sin mucho gasto telefónico, esto no será necesario. El servidor web Apache (tanto para Windows como para Linux) ya viene con el intérprete de Perl 5 por defecto.

Para realizar los scripts, que como dijimos anteriormente son en ASCII, se puede utilizar WordPad, un editor común, o **UltraEdit-32** (**www.ultraedit.com**), una gran herramienta a la hora de programar varias líneas de código (**Figura 3**). Este último tiene opciones que resultan de gran ayuda a la hora de verificar si lo que hicimos está bien.

Primeros pasos

Para correr los ejemplos, con Active Perl instalado en la máquina, ejecutamos la siguiente sentencia en una ventana de MS-DOS, detrás del símbolo de sistema:

```
C:\pruebas>perl archivo.pl
```

donde **perl** es el intérprete del lenguaje (perl.exe), y **archivo.pl** es el script que creamos.

En el caso de UNIX hay que indicarle, **siempre** dentro del mismo archivo, el directorio donde está alojado el binario de Perl: en la mayoría de los casos, **/usr/bin**. Luego, para que corra, debemos cambiarle las propiedades, de modo que sea ejecutable:

```
>chmod 755 archivo.pl
```

Este paso debe realizarse una sola vez por cada archivo nuevo (en general se hace en la primera ejecución).

Finalmente, para correrlo sólo basta con invocar el archivo por su nombre:

```
>archivo.pl
```

Analogías con C/C++

Teniendo en cuenta un par de detalles, para los programadores acostumbrados a trabajar en C, o incluso para los que recibieron nociones básicas de ese lenguaje, la programación en Perl les resultará cómoda y sencilla.

Uno de esos detalles es la forma de declarar las variables. Para ello se les antepone un símbolo según su naturaleza, por ejemplo, para una variable **escalar del tipo entero** sería **`$valor, $a, $b,`** etc. Para un vector, en cambio, se le antepone el símbolo arroba, por ejemplo: **`@datos1[0]`**. La ventaja de esto reside en que no debemos preocuparnos por su declaración, ya que el intérprete se encarga de acomodárselo según su conveniencia (ver **Tabla 1, Clases y tipos de variables**).

Veamos un ejemplo:

```
$dato="6547CNN";
```

Para saber más

FreeCode.com	www.freecode.com/cgi-bin/search.pl?query=perl
ActiveState.com	www.activestate.com
Perl.com, mucha info sobre este lenguaje	www.perl.com
Más material para quemarse el bocho	language.perl.com
PerlClinic.com, muy recomendado	www.perlclinic.com

crea la variable escalar **dato** y le asigna el valor o cadena **6547CNN**. Ahora, si imprimimos:

```
print $dato;
```

obtendremos como resultado: **6547CNN**. Pero, si en cambio queremos sumar **dato + 1**:

```
print $dato+1;
```

el resultado será el siguiente: **6548**, sin que esto modifique el contenido del escalar **dato**. Perl toma todos los caracteres iniciales que formen un número correcto, y ese número es el que interpreta.

La idea de Perl es simplificar las cosas, pero debemos saber bien cómo es que lo hace para no tener problemas.

Más similitudes

Las conocidas herramientas **for**, **while** e **if** de C también funcionan en Perl; algunas cambian muy levemente la sintaxis, pero el concepto es exactamente el mismo.

Los vectores se manejan de la misma forma, pero un dato interesante es que en Perl, para contar los elementos contenidos por un vector sólo hace falta la siguiente sentencia:

```
@vector = ( 100, 102, 103, 104);
#una rápida forma de cargar los datos.
$cant_elem = @vector;
        #una más rápida forma de contar cuántos elementos hay.
```

El valor de **$vector[2]** será 103 y el de **$cant_elem** es 4.

Manos a la obra

El proyecto completo que realizaremos será un contador con estadísticas; algo parecido al que solemos usar en las páginas web, como el de **TheCounter.com** o el de **Ciberstats.com**. El nuestro será un ejemplo rápido, el cual podrá perfeccionarse y completarse con más investigación.

La idea es crear una base de datos de usuarios, quienes emplearán el contador. Cada uno tendrá una cuenta, que se irá incrementando según la cantidad de visitas que reciban. Este contador aumentará el número de visitas sólo con la carga completa de una imagen. Esta opción la podemos usar con nuestras páginas web, con amigos o con abonados, mejora de por medio, si tenemos un servidor de Internet.

El proyecto completo consta de tres módulos. El primero crea la base de datos y agrega nuevos usuarios (**Figuras 1** y **2**). El segundo es el contador propiamente dicho, el cual almacena las visitas y una estadística sencilla del navegador que estaba usando el internauta que entró en la página. El tercero y último es el que presenta el informe de toda la data recolectada.

Veamos el primer ejemplo, mientras avanzamos en la explicación. Lo primero que haremos es pasarle la información a nuestro script de Perl mediante un formulario (**formulario.html**) con el método GET. Esto nos permitirá ver en la barra de navegación cómo es enviada. Encontrarán todo el código completo en la carpeta **WEBMASTERS** del CD:

```
<form name="form1" method="get" action="../cgi-bin/agregar.pl">
```

Tabla 1. Clases y tipos de variables

En esta tabla podemos ver cómo definir las variables para los distintos tipos de datos.

Clase	Símbolo	Tipos
Escalar	$	Entero, Real, Cadena, Puntero
Vector	@	Vector de escalares
Hash	%	Vector asociativo de escalares
Archivo	(ninguno)	Identificador de archivo

El código

Sin excepción, en la primera línea debemos informar dónde está ubicado el binario del intérprete de Perl. Esto se debe hacer siempre a continuación de los símbolos #!, los cuales no son reconocidos por Perl y sí por UNIX. En la mayoría de los servidores el camino es **/usr/bin**, de lo contrario deberán consultar al administrador.

La siguiente línea es la ubicación, también en el servidor, pero esta vez de algo conocido: nuestra base de datos. Este archivo será un ASCII almacenado en la carpeta que indiquemos. Por ejemplo: el directorio **/datos** dentro de **/public** en el directorio raíz del espacio en el servidor. Este archivo aún no existe, y será creado la primera vez que agreguemos un usuario.

Luego invocamos una a una las distintas subrutinas, las que serán declaradas a continuación del programa principal (dentro del mismo archivo .pl). El símbolo **&** delante de cada subrutina es el equivalente al **$** para los escalares y **@** para los vectores. Esto le indica a Perl que debe ejecutar esa subrutina, la cual debe ser declarada más adelante. Fíjense que en la declaración no lleva el símbolo **&**, sino que se le antepone la palabra **sub**.

La primera de estas subrutinas es *&parseform*; tan complicada de explicar como necesaria. Por ahora recomiendo que la copien tal cual está cada vez que la necesiten. La forma como lo hace excede a nuestro primer acercamiento de esta nota y dejaremos esta explicación para más adelante; pero lo que hace es sencillo. Es una de las piezas fundamentales: la rutina encargada de analizar el formulario HTML que llama a nuestro cgi, y de adaptar los datos ingresados (tanto en modo GET como en POST) pa-

Lugares que corren Perl

Para muestra basta un botón, así que no dejen de visitar alguno de los sitios que basan su funcionamiento en Perl.

Terra, el famoso portal, utiliza este lenguaje	www.terra.com.ar
VideoMovil, el primer sitio de alquileres online	www.videomovil.com.ar
Otro sitio que funciona gracias a este lenguaje	use.perl.org

LISTADO 1 - Agregar. pl

```
#!/usr/bin/perl

$BaseDatos = "/server/www/www.midominio.com.ar/public-
/datos/usuarios.txt";

# COMIENZO DEL PROGRAMA
# de la siguiente forma invoco las diferentes subruti-
nas o funciones.
&parseform;   #analiza el formulario
&ObtenerInfo;  #recupera la información anterior
&Agregar;    #agrega el usuario a la base de datos
&Salida;    #Muestra una pantalla de salida

# FIN DEL PROGRAMA

# Obtengo la información del pedido
sub ObtenerInfo
{
 $nombre = $FORM{'nombre'};
 $email = $FORM{'email'};
 $tipo  = $FORM{'tipo'};
}

#Agrego el nuevo usuario
sub Agregar
{
  open(USUARIOS,  ">>$BaseDatos")  ||  open(USUARIOS,
">$BaseDatos");
 print USUARIOS "0|$nombre|$email|$tipo\n";
 close (USUARIOS);
}

sub Salida
{
 print "Content-type: text/html\n\n";
 print <<HTML; #Imprime todo lo que sigue a continua-
ción (por ejemplo el HTML de salida) hasta que encuen-
tra la cadena "ETIQUETA"

<h1>Base de datos</h1><hr>
<p>Ingreso del usuario satisfactorio:</p>
<table width="95%" border="0" cellspacing="1" cellpad-
ding="1" align="center">
  <tr>
   <td>
    <div align="right">Nombre</div>
   </td>
   <td>
    <b>$nombre</b>
   </td>
  </tr>
  <tr>
   <td>
    <div align="right">e-mail</div>
   </td>
   <td>
    <b>$email</b>
   </td>
  </tr>
  <tr>
   <td>
    <div align="right">Tipo de contador</div>
   </td>
   <td>
    <b>$tipo</b>
  </tr>
 </table>

HTML
#IMPORTANTE: Debe estar escrita pegada al margen y sin
nada a la derecha
# de lo contrario Perl no la reconoce.
}

# Subrutina Parseform (Análisis de formularios).
# Uso: copiá y pegá esta función dentro de tu script
CGI cuando necesités recolectar información de un for-
mulario, luego invocá &parseform;
# La data es almacenada en el vector asociativo %FORM.
Para acceder usá la forma $FORM{'nombre_del_campo'}.
sub parseform
{
   if ($ENV{'REQUEST_METHOD'} eq 'GET') {@pairs =
split(/&/, $ENV{'QUERY_STRING'});}
  elsif ($ENV{'REQUEST_METHOD'} eq 'POST')
    {
    read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
    @pairs = split(/&/, $buffer);
    }
  else
    {
    print "Content-Type: text/html\n\n";
      print "<HTML><h2>Bad or unknown request met-
hod.</h2></HTML>";
    exit(0);
    }
  foreach $pair (@pairs)
    {
    local($name, $value) = split(/=/, $pair);
    $name =~ tr/+/ /;
    $name =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", he-
x($1))/eg;
    $value =~ tr/+/ /;
   $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", he-
x($1))/eg;
    $value =~ s/<!-(.|\n)*->//g;
    $FORM{$name} = $value;
    }
}
```

ra que sea comprendido por nuestro programa. En pocas palabras: almacena los valores ingresados separados por los nombres que se les asignaron en los campos de los formularios. En nuestro caso son **$FORM{'nombre'}**, para extraer el dato ingresado en el campo llamado **nombre**; **$FORM{'email'}**, para el campo **email**, y así los demás.

Una vez analizado el formulario, estamos en condiciones de tomar esa información asignándola a variables escalares comunes. Esto lo hacemos en la segunda rutina llamada **&ObtenerInfo;**, que nada más asigna los campos anteriormente analizados en las nuevas variables escalares.

La tercera rutina es **&agregar**, encargada de adicionar todos los datos obtenidos al final de la base de datos de usuarios del sistema.

Lo que vemos son dos **Opens** separados por el operador **||** (*doble pipe*), que se encarga de decirle al intérprete de Perl que si falla la primera función, es decir, si da algún error, ejecute la segunda. Entonces intenta abrir el archivo en modo *agregar al final* (operador >> delante del nombre del archivo). Si el archivo no existe, crea uno nuevo, vacío, listo para ingresar información (operador >).

El nombre del archivo fue dado al comienzo e ingresado en la variable **$BaseDatos**. También, dentro de la función **open**, le asignamos una etiqueta que identifique a este archivo abierto, para poder manejarlo y direccionarle la información. Inmediatamente después agregamos la línea que contiene la información del usuario de un modo sencillo: imprimimos en el archivo identificado con el nombre **USUARIOS** la línea que contiene los datos que ingresamos en el formulario.

La base de datos será un archivo ASCII que contendrá los distintos campos separados por el signo **|** (*pipe*); elegido arbitrariamente. Este signo podría haber sido cualquiera, lo importante es usar uno que sepamos que nunca será ingresado en el formulario. En otra ocasión veremos cómo identificarlo. Es importante que no tenga retornos de carro (**<ENTER>** o **\n**) en medio de cada línea. El cero inicial es el contador listo para usar y, a continuación, le siguen los demás campos. Un ejemplo de base de datos sería el siguiente:

```
0|Martín|martins@hotmail.com|1
0|Pikachu|pikachu@pokemon.com|1
0|Andrea|andreag@yahoo.com|2
```

El último campo es el que usaremos para saber las preferencias del usuario, si queremos que el contador sea exhibido, o no, que simplemente figure una publicidad o un logo.

Por último, la presentación final, utilizada para transmitirle al usuario que el programa se ejecutó normalmente.

Otra de las sentencias clásicas de Perl, obligada si queremos enviar texto a la pantalla del navegador, es **print "Content-type: text/html\n\n";**. Debemos avisarle al browser lo que vamos a enviarle a continuación. Inmediatamente después, estamos listos para pasarle el código HTML que queremos mostrar, y la forma más sencilla de hacerlo es con etiquetas.

Con la sentencia **print <<HTML;** decimos que imprima todo lo que sigue debajo (incluso líneas comentadas con **#**), hasta que encuentre la etiqueta con el nombre que le asignamos, por ejemplo **HTML**. Esta palabra debe escribirse en el margen izquierdo sin ningún espacio, ni delante ni detrás, y con un **<ENTER>** para finalizar (tampoco lleva el punto y coma).

Dónde aprender Perl

Hasta hace muy poco tiempo, era muy difícil aprender un nuevo lenguaje de programación por cuenta propia. Los textos o libros generalmente eran difíciles de conseguir y, en la mayoría de los casos, venían en inglés, lo cual los limitaba solamente a un grupo reducido de personas.

Afortunadamente, con la llegada de Internet, aparecieron las páginas personales y se popularizaron los tutoriales y demás sitios que ofrecen herramientas, cursos y tutoriales en línea.

Es así que ahora es muy fácil encontrar lo que buscamos en cuestión de minutos, podemos imprimirlo y leerlo offline en el lugar en que estemos.

Si vamos a comentar algunos de los mejores sitios para aprender este lenguaje, no podemos olvidarnos de **Perl.org** (**www.perl.org**), el sitio oficial, donde hallaremos mucha información. No está en castellano pero, para los que dominan el idioma inglés, les será de gran ayuda.

Si visitan **Los Tutoriales** (**www.lostutoriales.com**) o **Programación.Net** (**www.programacion.net**), ambos comentados en la nota de tapa de esta revista, podrán encontrar una gran cantidad de recursos gratuitos y tutoriales, sobre este y otros lenguajes similares. Otro sitio parecido que les será de gran ayuda es **Webexperto** (**www.webexperto.com.ar**). Si bien no tiene una gran cantidad de links, seguro que descubrirán algo de utilidad.

Finalmente

Como vimos hasta aquí, Perl se convierte en una opción más que interesante y, sobre todo, económica para construir un sitio con contenido dinámico e interactivo. Además, gracias a los servidores web como Apache, tanto para Windows, como para Linux, podemos mantenerlo fácilmente en nuestra propia PC sin necesidad de subirlo al servidor cada vez que queremos ver algún cambio o probar algo.

Tampoco nos olvidemos de que, por sobre todas las cosas, es una alternativa muy económica y que no requerirá de inversiones costosas en cuanto a sistemas operativos y hardware.

Dónde alojar Perl

Éstos son sólo algunos de los sitios que ofrecen soporte para alojar páginas desarrolladas en Perl.

Rec-Inf, recursos informáticos	**www.rec-inf.com.ar**
Adn-Net	**www.adn-net.com.ar/hosting.html**
Rayp.com	**www.rayp.com/hosting.htm**
Neco.com.ar, Web hosting	**www.neco.com.ar/webpoint/hosting.html**